

DMDC QA TESTING GUIDELINES

v. 0.8

Table of Contents

(NOTE: Use CTRL-click to jump to a location.)

1.0 Purpose	4
1.1 Scope	4
1.2 DEERS Analysis	4
2.0 Testing Overview	4
3.0 Common Data Scenarios Needing Testing	5
3.1 Identifiers	5
3.2 Types of persons	5
3.3 Person in multiple (two+) families	5
3.4 Names	5
3.5 Dates	6
3.6 Data of Birth (DOB) / Age / Death Date	6
3.7 Data segments (PNL, PNLEC, Enrollment, etc...)	6
3.8 Enrollment data	6
3.9 'Numbers' as strings	7
3.10 Soft Delete (Cancelled)	7
3.11 Gender	7
3.12 Identity	7
4.0 Application Versioning	7
5.0 Test Data Management	7
5.1 DOOR: Tracking Data Usage in Model Office	8
5.2 Finding Data to match scenarios	8
5.3 Creating test data	8
5.3.1 Data creation tips and tricks	9
5.4 Test data repository	9
5.5 Initializing Test data	9
6.0 Test Heuristics	10
6.1 Boundary value Analysis	10
6.1.1 Misc Boundaries at DMDC	10
6.2 Equivalence Class Analysis	10
6.2.1 Equivalence at DMDC	11
6.3 All-pairs analysis	11
6.3.1 Pair analysis at DMDC	11
6.4 Misc application scenarios	11
7.0 Test Process	12
7.1 Get Involved Early	12
7.2 Have a Plan	12
7.3 Keep Good Notes / Testing Documentation	13
7.4 Identify Test Cases	13
7.5 Test Case Objectives	14
7.6 Review!	14
7.7 Flesh out Test Cases in a Test Suite / Matrix	14

7.8 Execute Tests	15
7.9 Report	16
7.10 Review	16
7.11 Test Maintenance	16
8.0 Issue / Bug / Defect / Ticket Tracking.....	16
8.1 “Your bug report is your representative”	16
8.2 Terminology “What is an Issue / Ticket”	16
8.3 Who enters Tickets.....	17
8.4 Top Tips.....	17
8.5 Notifications	18
8.6 Issue Lifecycle “A Bugs Life”	18
8.7 Tracking Data Model Issues	19
9.0 Automation	20
9.1 Deciding what (and how) to automate	20
9.2 Different Types of Automation	20
9.2.1 Batch test automation	20
9.2.2 TestPartner Test Automation	21
9.2.3 Miscellaneous Tools / Processes	21
9.2.4 Additional Info	21
9.3 “Oracles” to use in Test Automation	21
10.0 Monitoring Testing (AppMonitor)	22
11.0 Security Testing	23
11.1 Authentication, Authorization and Password Changes	23
12.0 Testing as Part of a System (“End to End”)	23
12.1 Auditing & RUN / SUBM ID’s	23
12.2 Logging	23
12.3 Notifications.....	23
12.4 Triggers	23
13.0 SQL	23
13.1 Some Wildcards to use with “LIKE”	24
13.2 Common SQL Functions / Commands	24
13.3 SQL *Plus	25
13.3.1 SQL *Plus Formatting	25
13.4 Trigger / Procedure Testing	26
14.0 Oracle Data Dictionary	26
15.0 Potential Future Topics	26
Appendix A: Glossary	26

1.0 Purpose

The purpose of this document is to help the QA team become more effective:: find and report more issues and do so more quickly and efficiently. To achieve this goal, QA must share its methods for best practices and standards so that communication and test efficiency can be improved.

1.1 Scope

This document provides testers with guidelines for testing applications. The document covers the general processes and templates that are used within the West Coast (WC) testing group. The focus is primarily on functional testing.

This initial version is focused on capturing and refining existing processes. Over time this process will progress and be used to identify and capture additional areas where improvement may be necessary. These areas may include additional testing practices and testing tools and resources used in performing application development testing.

This document does not focus on configuration testing, conversion testing, data integrity testing, fallback testing, portability testing, regression testing, security testing, or other more specific testing at this time.

1.2 DEERS Analysis

A recent short survey was performed within DEERS, the results of are as follows:

(Note that these results may apply to testing in both development and QA)

1. Very little, or no, planning for testing throughout the application development life cycle.
2. We cannot measure the effectiveness of testing.
3. What needs to be tested?
4. Heavy reliance on QA for quality of software.
5. There are different testing processes or procedures. The best testing processes are not shared.

2.0 Testing Overview

Software testing is the process used to help identify the correctness, completeness, security and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors.

More 'theory' on testing can be found at http://en.wikipedia.org/wiki/Software_testing

Other items to work in:

- Psychology matters – testers find more issues if looking to find issues as opposed to looking to prove that an application works. (Chapter 2, "The Art of Software Testing, Second Edition")
- The "economics" of testing: "Given constraints on time and cost, the key issue of testing becomes: **What Subset of all possible test cases has the highest probability of detecting the most errors?"** -- (Chapter 2, "The Art of Software Testing, Second Edition")
- Where test case selection comes in:
 - Use Equivalency Classes: identify the classes of test cases (with intent to only run one or two cases within the classes)
 - Boundary Analysis: many issues occur on the edge

3.0 Common Data Scenarios Needing Testing

The below list identifies a subset of common data scenarios that may need testing in a given application (not all items are applicable to all applications.) Many of these items have been identified via bugs found in existing products.

3.1 Identifiers

1. Using different ID types, especially for IDs in different tables – PN_ID and DOD_EDID_PN_ID are most commonly used. Reason: have had bugs where data, such as OGP, is returned when pulling with an identifier from PN_XR, but doesn't work when pulling via DOD_EDID_PN_ID or HICN. Identifiers:
 - a. DBN (DoD Benefit Number) (DEERS_FAM_ID + DEERS_BNFRY_ID, in MD_BNF)
 - b. HCN (Health Care Number) – An external name for the DEERS_FAM_ID (see below)
 - c. DEERS_FAM_ID (MD_BNF if primary source; also in BNF_SAT)
 - d. DMDC_ID
 - e. DOD_EDID_PN_ID (also called 'PTNT_ID', in EDI_SAT, and sometimes EDI_CFN) NOTE: Max value supported by the 'core': 2,147,483,647 ($2^{31}-1$)
 - f. HICN (Health Insurance Carrier Number? in HIC table); supported by a limited number of applications
 - g. PN_ID (SSN, FIN, ITIN, TIN, ...) (in PN_XR & SF_XR for sponsors)
2. IDs (SSN, FIN, etc) with leading 0's
 - a. (some developers have treated SSN as a number, causing applications to fail when used with IDs having one or two leading 0's)
3. Persons having old and new identifiers (PN_ID, DOD_EDID_PN_ID) – inquiries using old & new identifier (some applications return the identifier passed in, others return the current identifier)
4. Person with two SSNs
5. Identifier 'valid' but type is not (passing DFI, but saying it is an SSN)
6. Pulling a dependent when looking for a sponsor and pulling a sponsor when expecting a dependent (within same family, different family)
7. Also see 'Dual Eligibility' section.

3.2 Types of persons

1. Sponsor
2. Dependent (spouse, child, parent, ...)
3. Newborn (yes, a special category called 'Newborn')
4. Non-affiliated
5. VA (?)
6. Other?

3.3 Person in multiple (two+) families

1. Child of a service member joins the service (before and after eligibility from the parent goes away)
2. Service member marries a service member
3. Child of two service members
4. Person is former spouse of one service member and the current spouse of another service member

3.4 Names

1. Special characters (space, ' , -)
2. Null first name, Middle name, cadency name (yes, first name is not required)
3. Short name (1 or 2 characters)
4. Max length name (per PN table)
5. Mixed capitalization (McAndrew, etc)
6. Suspense vs non-suspense values
7. Use of aliases (AN table)
8. Leading/trailing white space should be trimmed (spaces, tabs, etc)

9. Leading numbers should be removed from first, middle, last
10. Data in SUSP fields
11. Display data containing numbers

3.5 Dates

1. Months with single digit and two digits; leading 0's
2. First day of month, last day of month, leap date

3.6 Data of Birth (DOB) / Age / Death Date

1. Person over 100; born in 1800's
2. Under / over (some medical/dental rules: 4 (dental), 21, 23, 26, 65
3. Person with null birthdate (yes, this happens in our database)
4. Suspense birth and death dates
5. Pulls before birth and after death

3.7 Data segments (PNL, PNLEC, Enrollment, etc...)

1. No segments in a given table (where allowed)
2. Single segment
3. Historic / Current / Future segments
4. Segment lengths (multi month, multi-year, less than month, single day)
5. Segment start/end dates (begin, mid, end of month – month with 30, 31, or 28/29 days)
6. Multiple Non-overlapping (continuous vs non-continuous)
7. Multiple Overlapping segments -- Examples include:
 - a. PNL: Civilian/Contractor who is a retired service member
 - b. PNL: Civilian/Contractor who is in the Guard / Reserve
 - c. PNLEC/PNL : Transitional Assistance (PNLEC) overlapping PNL (such as Retiree, Reserve/Guard, civilian, etc)
 - d. NOTE : PNL/PNLEC overlap rules in the c_spon_coll62 table of PDR
 - e. See enrollment section for enrollment overlaps
8. Data changes within a given segment. For example:
 - a. Pay Grade (PG) or Unit (UIC) changing during a personnel (PNL) segment
 - b. Various activations (PNLEC) within a given PNL segment

3.8 Enrollment data

1. Different types of enrollments
 - a. Medical, Dental, Special
2. Different subclasses of enrollments: Medical
 - a. Eligibility from ASG_HCDP, have PCM
 - i. Standard, PRIME
 - b. Eligibility from MED_ELIG, do NOT have PCM, plus other rules
 - i. TRS (TRICARE Reserve Select)
 - ii. TRR (TRICARE Reserve Retired)
 - c. Eligibility from MED_ELIG, some do/don't have PCM, plus other rules
 - i. TYA (TRICARE Young Adult)
3. Fee vs Non Fee paying plans
4. '011'/'013' HCDP_PLN_CVG_CDs can overlap with other enrollments (HCDP / MHCC / UMHCC / etc, tables)
5. Multiple enrollments per policy
6. Multiple contractors (EMC) or PCMs (TPCM_SLCT) per enrollment
7. Transitions from one enrollment type to another
8. Segment length

3.9 'Numbers' as strings

1. For any 'Number' being stored as a string (PN_ID), test with leading 0's to ensure that the code does not use as a number and trim leading 0's

3.10 Soft Delete (Cancelled)

1. 'Cancelled' segments: segments with a TRSN_CD of 'F', 'E', and '4' should NOT be returned/used by most applications

3.11 Gender

1. Null gender
2. Changed gender
3. Suspense gender
4. ALIAS_SEX table

3.12 Identity

1. TBD
2. Current vs expired vs revoked
3. None, single, multiple
4. CAC vs PIV
5. Token Characteristics
6. ...

4.0 Application Versioning

A few brief thoughts on versioning. Some might be obvious, but....

- Every build to QA should have a unique version number
- "It would be nice"™ for every product release track to have a unique portion of the version number. Aka, when talking about the a release of an application such as GIQD, can refer to that release as the 4.1 release, without having to reference 4.1.00.00.3 (or whatever), which changes in every build to QA.

An implementation of this style of version is the standard for Identity Web Services (IWS) applications, and is documented at

<http://teamsites.ds.dhra.osd.mil/teams/devel/LegacyTwiki/IWSVersioningScheme.html>

5.0 Test Data Management

An issue that QA encounters while testing is its ability to maintain a "clean" set of test data. In this context a clean set of test data would be: A predetermined set of data that is purposely in a known state. This ideal has not always been achieved as test data 'ages out' and the databases are shared by multiple QA testers, Research, and others.

In order to maintain a clean set of test data QA testers should observe the following:

- You should change only the data you "own" in Model Office (see the DOOR tool for information on ownership tracking)
- If you do not currently own the required data in Model office then create the data either by cloning an existing record using DataCopy, creating new data using tools such as RAPIDS and WebDOES, or find a record in production and copy into the test region.

- If testing a read only application, you may be able to leverage test data already in use by other read-only applications (e.g. HCCMTF, Claims, etc.). Contact the primary tester of the read-only application for information on test data, check DOOR, OR locate the data in Model Office using SQL.
 - See the Testing Assignments Sheet for information on an application's primary tester.
 - Some SQL used in the past to find test cases can be found at
K:\Qa_Testing\SQLscripts\research\test_case_lookup

(Note: Some data may already be in use in an update application which will change the data)

5.1 DOOR: Tracking Data Usage in Model Office

As stated above, QA has one set of databases shared by the entire QA organization as well as Research, Data Quality, etc. These databases are used for:

- Standard QA Testing
- Regression Testing
- Automated Testing using "Gold File" result analysis/comparison

Before using any data for testing it is important to check ownership of that data using DOOR, the Data Ownership Operational Repository. It is a database that tracks which applications use testing data in Model Office on an ongoing basis, with the goal of minimizing time-wasting collisions.

Basic usage principles:

- You claim a person for an application you test, not for yourself.
- You can only claim sponsors that already exist in the PDR database.
- Claiming a sponsor 'claims' the whole family.

Additional information on DOOR is available from SharePoint under <http://teamsites.ds.dhra.osd.mil/teams/es/qa/Tools/default.aspx>. The DOOR database can also be checked using SQL. See K:\Qa_Testing\SQLscripts_shared_readme_SQL_overview.doc for info on using DOOR (and other shared SQL files)

If/when that data was last modified, use the 'Run Audit' tool (from 'DMDC Tools') to determine who, if anyone, has made a change to the data using one of the DMDC tools (this tool cannot detect if changes were made using SQL.)

5.2 Finding Data to match scenarios

- Some data defined with test cases (especially in automation)
- Search the database (custom SQL or previously created SQL, such as can be found at
K:\Qa_Testing\SQLscripts\research\test_case_lookup)

NOTE: check DOOR (see above) before modifying data

5.3 Creating test data

There are various methods for creating test data in QA testing environments. For Model Office:

- Clone existing data in Model Office using DataCopy
- Copy data from Production or contractor test using DataCopy
- Create using RAPIDS (see your lead for info on using RAPIDS)
- Create / update / change using the 'QA Tools' application (see the primary tester / your lead)
- Create / change using SQL (requires deep knowledge of the database)

- Partial data sets using SQL to create data initialization SQL (for example, SQL is available to create SQL that re-creates enrollment data -- See K:\Qa_Testing\SQLscripts_shared_readme_case_creation.doc for info on this (and other shared SQL files))

5.3.1 Data creation tips and tricks

- When creating a person with photo image data, you need to manually put records in the e2r2.PHOTO table for that person. There's no synonym in ADW for this table. Then you can run SQL scripts against ADW that will insert/update/delete the PHOTO_IMG table that will be returned by the record generator.

5.4 Test data repository

There is a 'DataRepository' for test data in Model Office (plus a different one for DBIDS NextGen and one in 'Test' for developers to use.) It is a bucket to hold data in a known state, in a database that is NOT used for actual testing. It is used to ensure that you can get your data back into a known good data state.

When you have your data (claimed in DOOR) in a good initial state, copy the data (DataCopy) into the DataRepository for future use. Information on DataCopy is available at <http://teamsites.ds.dhra.osd.mil/teams/es/datasvcs/datacopy/SitePages/Home.aspx> - please see your lead type(s) if you have additional questions.

5.5 Initializing Test data

For repeatable testing, it is important that your test data is in a known state that support the scenario being testing. A number of methods are used to ensure that your test data is in a known good state.

Regardless of the method used, it is good to ensure (explicitly check) that your data is in a good state. For example, if you need to ensure that derived data isn't created for a specific scenario, make sure the scenario actually exists!

Methods of ensuring that data is in a known initial state:

- GUI DataCopy
- Command-line DataCopy (usable from Automation, such as PFT)
- DBIDS DataCopy (used in DBIDS NextGen)
- Person/family deletion using DataCopy (used in PFT and other automation)
- Enrollment re-creation SQL (used in MHCDP, DHCDP, NtfyD, and other automation)
- Custom SQL for data creation (NOT recommended for PDR/MedSat; used for ADR/ADW)
- SQL for data updates (used in multiple applications)
- No initialize, just check (only for read-only applications; can be faster than explicitly initializing data on ever test run; used in automation such as HCCI and Claims Coverage)

6.0 Test Heuristics

Lesson 38 from “Lessons Learned in Software Testing”: Use heuristics to quickly generate ideas for tests.

- “A *heuristic* is a way of making an educated guess”
- “*Test at the boundaries*”
- “*Test every error message*”
- “*Run tests that are annoying to setup*”
- “*Avoid redundant tests*” (*equivalency analysis, all-pairs*)

6.1 Boundary value Analysis

From Wikipedia: http://en.wikipedia.org/wiki/Boundary_value_analysis

“Boundary value analysis is a software testing design technique to determine test cases covering off-by-one errors. The boundaries of software component input ranges are areas of frequent problems. ”

- High & low boundaries
- Min-1, min, min+1
- Max-1, max, max+1
- Segments: 0, 1, 2, max (99)
- External & internal boundaries
- Bytes: 64, 128
- Size of pools (DB connections, sessions)

6.1.1 Misc Boundaries at DMDC

- Activation dates – start, end, change
- 180 days after projected end of ‘On AD’ condition
- Child’s ages: 4, 21, 23, 26
- Death dates, death+180, death+3Y, last day of month of death
- Plan start dates
- Dates to/from college
- 0 segments, 1 segment, many segments
- Enrollment start / end dates (day of, day before, day after)
- 90 day historic enrollments
- 30/60/90/180 day future segments

6.2 Equivalence Class Analysis

Wikipedia: http://en.wikipedia.org/wiki/Equivalence_partitioning

“Equivalence partitioning is a software testing related technique with the goals:

To reduce the number of test cases to a necessary minimum.

To select the right test cases to cover all possible scenarios. ”

For example, the number of a month (1-12)

- 1-12 are in the ‘valid’ partition
- <1 are in invalid partition 1
- >12 are in invalid partition 2

NOTE: that 1-9 and 10-12, while both in the ‘Valid’ partition, they are different lengths and generally should ensure that dates are tested with 1 and 2 digit months

More info: http://en.wikipedia.org/wiki/Equivalence_partitioning

6.2.1 Equivalence at DMDC

Different items impact different apps, so not a blanket statement, but some items that are equivalent in some applications....

- Various branches (Army, Air Force, etc in Active Duty, Reserve, Retired, etc.)
 - All 'Active Duty', regardless of branch, tend to get the same benefits
 - Tend to get the same medical benefits
- Sponsors have certain characteristics regardless of service status
- Dependents (for some usages; spouse vs child vs parent have different benefit sets)
- Guard and Reserve are quite similar
- Different groups for different age of children: Child under <4, 4-21, 21-23 in college, >23, incapacitated

6.3 All-pairs analysis

From Wikipedia: "All-pairs testing or pairwise testing is a combinatorial software testing method that, for each pair of input parameters to a system (typically, a software algorithm) tests all possible discrete combinations of those parameters."

- Combinatorics are the enemy – can't test all combinations
- Each test should cover multiple parameters
- Part of lesson 54 in 'Lessons Learned'

Info on wiki: http://en.wikipedia.org/wiki/All-pairs_testing

6.3.1 Pair analysis at DMDC...

- Various branches within group (Army, Air Force, etc in Active Duty, Reserve, Retired, etc.)
- Sponsors & dependents (deps depend on app....)
- Guard and Reserve are quite similar
- Child under 4, 4-21, 21-23 in college, between 21/23 and 26 (TYA eligibility, depending on college and other exceptions), incapacitated
- Activation dates – start, end, change
- 180 days after projected end of 'On AD' condition
- Death dates, death+180, death+3Y, last day of month of death
- Plan start dates
- Dates to/from college
- 0 segments, 1 segment, many segments

6.4 Misc application scenarios

- System to system applications
 - All transaction types
 - All response types
-

7.0 Test Process

We work in a very dynamic environment, where schedules and specifications may not be what we would prefer. Push back where it makes sense, but we are not the Client. Give honest assessments, highlight risks, and, um, TEST!

Conversely, being under the DOD, we are audited once a year, which requires a certain audit trail to validate the testing that was performed. At a high level, the auditors want proof that everything released to production was tested. For example, we need to show:

- What was tested for a given run (test cases run, test case results.)
- That the code and tests for that release are in Version Control
- That issues are formally tracked

So, need to be agile with good process.

General steps in the testing process are described in the following sections.

7.1 Get Involved Early

1. Review any specifications (CMS, emails, etc) as early as possible.

QA thinks about things differently and is good at finding items that are not fleshed out in the specifications and/or can see dependencies that are not obvious to others. NOTE starting on the Test Plan / Test Cases before the requirements are final can help flesh out requirements.

2. Provide comments on what changes may help make things easier to test. There is a document with general ideas at U:\Everyone\J2EE\QA\testability.

NOTE: what can and cannot be implemented really depends on the project and schedule.

3. Create any needed directories on k: to track information related to the application / change.

7.2 Have a Plan

Planning is the art and science of envisioning a desired future and laying out effective ways of bringing it about."

--Planning, MCDP5 U.S. Marine Corps

And to paraphrase, "No plan survives contact with the enemy, but better to modify an existing plan than winging it."

A good test plan lets you determine your method of attack and estimate the amount of work required for your project. A good test plan (hopefully) begins with a good functional and technical specification.

The test plan provides the following advantages:

- It will help you organize your thoughts on how to go about testing a software component effectively and efficiently.
- Others have an opportunity to assist and reinforce your test plan by providing comments on the plan.
- It will be very useful when you're ready to begin testing because you'll have something that you can refer to as you perform the testing. (a trail map for the daily tests and bugs, and a yardstick for measuring your progress.)

- It will be an excellent resource for future testers. Not only will it help them out if they're not familiar with a software component, but if they come to you asking how you tested a software component, you'll have something you can show them.
- Documenting all tests allows you to easily reuse and build on them in the future if needed, so your work is repeatable.

However, as useful as it is, a test plan is just that: a plan. It represents a testing strategy based on the best knowledge available at the outset of a project. As the test progresses, more will be learned about the software's strengths, weaknesses, and vulnerabilities. This should be anticipated and exploited. The key is to build flexibility into the test plan up front, so that adjustments can be made in real time as you learn and encounter development issues.

One item to remember: a Test Plan is both a product (client requirement) and a tool to test more efficiently.

Testing should not start until a test plan is created for a given release (unless approved by your Project Lead / Manager.)

System Test Plan Template on SharePoint at

<http://teamsites.ds.dhra.osd.mil/teams/es/qa/Reference/default.aspx>

7.3 Keep Good Notes / Testing Documentation

"There's no time to stop for gas, we're already late"

-- Karin Donker

In QA, we are not just consumers of project documentation (specs, etc), but are also document creators:

1. General notes / quirks / fleshing out of the specs: Even in the best of environments, specifications do not capture everything we desire as testers (For example, one estimate is that 80% of code is for handling error conditions. 80% of most specs does not cover error handling.)
2. We are required to maintain 'How to Test' documentation with the detailed steps on how to test an application (and good to have if you are the backup of an application!)

As you learn more about an application, keep information in a ???'qa_notes.doc'/Testing Procedures & Notes??? file in the 'specs' ('Test Docs?') directory for the project.

My general rule of thumb: if you ask the developer, Project Manager, QA Lead, etc a question about the application, if that information is not already in the spec or being added to the spec, add it into your document. You may forget the answer. The next tester will have the same question.

One item that is especially useful to keep as the tester is a section on "Data Validation" (see "K:\New DEERS\QA\X12\Test Plans\HCC for MTF Test Plan.doc" for an example.)

Your backup will thank you.

You will thank yourself when you come back to test that area three months (or three years) later, and have forgotten a detail or two....

7.4 Identify Test Cases

1. Start with an outline of the various equivalency cases and boundary conditions. Try to think of everything, even though you probably won't have time for everything. Sometimes, this will be a part of the test plan.

2. Provide more detailed test cases (in a new test suite/matrix or add to an existing one), with TC numbers, and high level initial conditions, steps, and expected results. Prioritize – scale of 1 – 5, 1 being highest: must, should, would be nice, probably not...

NOTE: provide detail for test cases that you think that you will have time to run.

Where possible, design with automation in mind.

Test Suite (Test Case Matrix) Template at
<http://teamsites.ds.dhra.osd.mil/teams/es/qa/Reference/default.aspx>

7.5 Test Case Objectives

Every test case needs to have a clear test objective (what/why are we testing)

- The highest concentration of analysis occurs during the development of test objectives. It is the requirement of each QA analyst to develop and demonstrate their aptitude for this analysis.
- Test Objects need to be understandable and reviewable by PMs, BAs, and other members of the project team. (aka, take it up a level from detailed technical speak)
- Treat test objective creation as specification testing. If you think that there is a problem with the specification, it is your responsibility to work with the BA to resolve the issue and to update the PM / Scrum Master when giving status updates on the Objective. Bottom line: the objectives need to be correct and in sync with the Functional Specification. If they are not, there is a bug in one or the other that must be resolved before the objectives can be finished.

Test steps need to include the initial state (app, data, environment)

7.6 Review!

Before you go too far down the testing road, have your test cases / outlines reviewed, both by an experienced tester and then provide an opportunity for the BA / developer to review.

7.7 Flesh out Test Cases in a Test Suite / Matrix

1. Create or find the data needed for the various test cases

NOTE: we have one shared database. Some of the test data is already spoken for. So, for update applications, it is best to create new data using RAPIDS or find a new person from production and copy into our test database (careful with the SQL use in production.)

Track your test data in DOOR as noted in the data tracking section

2. Create update / SQL needed for testing (validation SQL, changing data, etc.)
3. Provide detailed steps

It is recommended that you verify the initial condition of the data (using SQL*Plus and a 'master' SQL statement for your application.) We have a shared database, and this checks that the data is in the state expected by the test case. Your master SQL should check only the data needed by the given application. An example (from MHCDP) : K:\Qa_Testing\SQLscripts\asg_hcdp\mnt_aud2.sql (it takes a Sponsor DMDC_ID as a parameter.)

4. Where possible / appropriate, run new test cases before the new app or change comes to QA: prove that you can cause a failure so that you can prove that the new version of the application resolves the test case.
5. Automate where appropriate

7.8 Execute Tests

1. Promote software to QA: When an application is ready for testing, the developer will mark it as 'Ready for Movement' in CMS. When you are ready to test / done with the current version of the app in the given region, the QA Tester marks it as 'Ok for Movement.'
 - a. Model TWO is our primary test environment
 - b. Model ONE is used primarily when testing schema changes
2. Review claimed changes/fixes, your plan, etc.
3. Initial build validation: Run a subset of tests to get an overview of the release
 - a. Run your Acceptance Tests (small subset of total regression suite)
 - b. Validate Claimed Fixes (may do this before running acceptance tests, depending on the change/application)
 - c. Spot check changes
4. "Exploratory Testing" – Sometimes, the best way to learn an application / recent changes is to experiment with it while thinking about the requirements / end user expectations. Some call this 'Ad Hoc' testing, others 'Exploratory' (an intro at <http://www.satisfice.com/articles/et-article.pdf>.) If nothing else, it is a good practice to review your plan to highlight any holes.
5. Run a "full" test cycle

NOTE 1: definition of "full" depends on project, change complexity, available resources, and the existence of hard dates. As much as possible, if you think it won't be 'enough', discuss with your QA lead as soon as possible.

NOTE 2: when running tests, I like to do breadth before depth. Ensure all/most functional areas are working before checking every edge case in one area.

- a. Run new test cases. Analyze results.
 - b. Run regression tests. Analyze results.
6. Thoughts on results analysis:
 - a. If the actual initial condition does not match the expected initial condition (for example, someone killed the sponsor in your test family when the sponsor should be alive), try to figure out who changed, why, who "owns" (been using longer), and either reset to the expected condition or find new test data.
 - b. If issue, report (add to Tracker, plus keep a list in your QA directory for the application.)
 - c. If current results don't match the expected results, but current is ok, update the expected results.
 - d. Add or remove test cases as identified / needed
7. Track / keep the extended team in the loop
 - a. Keep summary of how many tests exist and their states (pass/fail/blocked/etc)
 - b. Let the developer know when testing is done, general status (hopefully can provide SOME good info), and a summary of the issues (point them to the issues list.) Plus other interested parties: test manager, project lead, etc, which varies greatly by product.
8. Repeat as needed for new builds.

7.9 Report

When testing is complete, we document the results in a 'QA SignOff' document and, for large releases, in a 'QA Test Report' (templates on the SharePoint QA templates site.) The QA SignOff provides a concise summary of the release, the testing that was done, information that the QA team has about the quality of the project, QA's recommendation on releasing the project, and documenting any concerns.

It helps ensure that all members of the project team know that the status of the application, and is used by QA during audits to help document the testing that was done for releases.

The QA SignOff needs to be sent for review at least one week before the product is released (a draft if testing continues after that point), and provided to the government product owner and your lead(s.) The SignOff is also included attached to releases within CMS.

7.10 Review

"Quis Custodiet Ipsos Custodes" (Who watches the watchmen?)

Ask your backup, team lead, or other team member to review your work; always good to have a second pair of eyes look at work product.

1. Review your QA notes – anything that you forgot to add / include?
2. Update acceptance tests? (new functionality? Old functionality no longer used? Better test cases?)
3. Any updates needed to 'How to Test'?

7.11 Test Maintenance

- Periodically (once a year, for example) review test suites / cases to determine which cases could be combined or removed without losing coverage.
- The test procedure document ('How to Test', 'How to Test') should be maintained once a year / with significant application changes

8.0 Issue / Bug / Defect / Ticket Tracking

Issues are tracked within Jira (<http://jira:8888>) Information on requesting an account and using Jira is at <http://teamsites.ds.dhra.osd.mil/teams/es/datasvcs/jira/SitePages/Home.aspx>

8.1 "Your bug report is your representative"

The above is Lesson #58 from "Lessons Learned in Software Testing" (Kaner, Bach, Petticord.)

Remember it. Live it. (And maybe read the entire chapter 4 on "Bug Advocacy")

The tickets that you create are your advocates for improving the quality of the product.

Make them count – clear, factual, and easy to understand. Why should I care?

Note: spelling and grammar count.

8.2 Terminology "What is an Issue / Ticket"

Definitions:

- Issue: Any aspect of the software that does not match with what is detailed in the software specification

- Ticket: Any item within Jira – could be an issue, enhancement, or task

8.3 Who enters Tickets

Tickets can be entered by anyone on the team, including:

- Quality Assurance
- Database Administrator
- External Project Management
- Development
- Tech Services
- Integration Team
- Business Analyst
- Internal Project Management
- etc

8.4 Top Tips

1. Detail all pertinent information that may be useful for diagnosing or reproducing the defect. Key components of a good ticket include:
 1. A clear, concise summary for the ticket. Written at a high-level, understandable by PMs, BAs, Developers, and QA. (“Summary” field in Jira)
 2. As needed, a more detailed description of the problem (“Description” field in Jira)
 3. Clear steps to reproduce, including data requirements (“Description” field in Jira; as appropriate, include impacted test cases in the “Test Case” field)
 4. The unexpected behavior seen (“Description” field in Jira)
 5. The behavior expected (“Description” field in Jira; as appropriate, reference documented requirements in the “Specification ID & Type” field)
2. Ensure “ancillary” fields are populated as well (“Affects Version/s”, “Component”, “Found By”, “Region”, “Test Type”)
 1. For issues that were discovered outside of QA, ensure that ‘Found By’ is set appropriately
3. A good tester will always try to reduce the ‘steps to reproduce’ to the *minimum necessary*. This is extremely helpful for the developer and other QA who have to reproduce the defect.
4. Always include in the ticket resolution / verification the unique build number of the version of software in which the ticket is fixed so that the person closing the defect can verify the resolution in the correct version of software.
5. All defects must go through the ticket tracking system. If you receive issue reports or resolutions via email and those reports or resolutions are not in the ticket tracking system, simply bounce the

emails back to the originator with a brief message: "Please put this into Jira. I can't keep track using email."

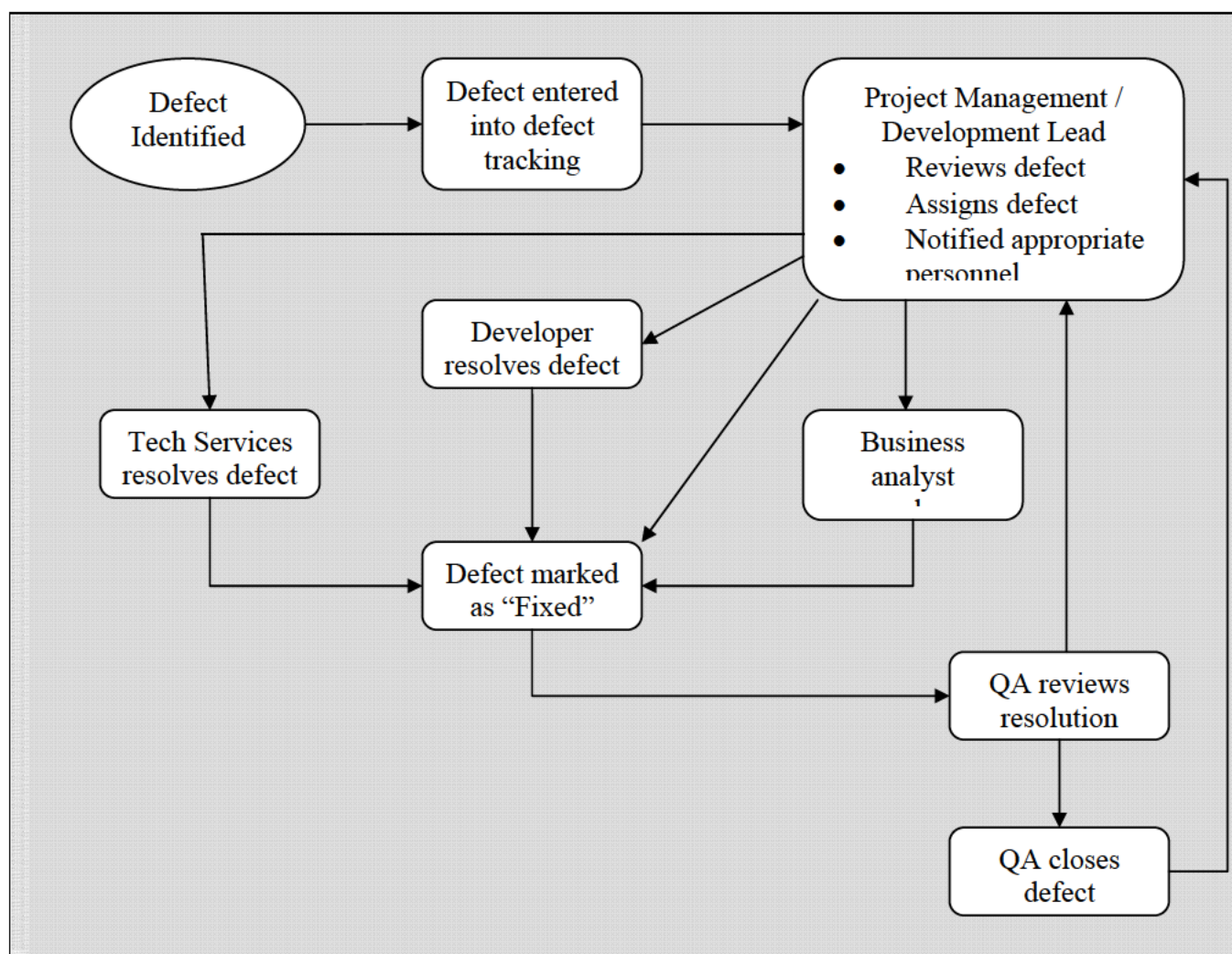
6. Always verify a defect can or cannot be reproduced before recording
7. Record the defect # in the "Comments" column of the QA Test Case Matrix for the associated Test Cases(s)

8.5 Notifications

TBD; generally automatic within Jira; something need to explicitly notify (email, phone) project team members

8.6 Issue Lifecycle “A Bugs Life”

1. Issue is identified
2. Ticket is entered into the defect tracking system by any individual with access
3. Item is automatically assigned as per the project rules in Jira (to PM, direct to developer, etc)
4. Project management (Role, not always someone with that title)
 - Reviews the ticket
 - Assigns the ticket to the appropriate personnel
 - Assigns the ticket to the appropriate release
5. Assignee resolves the ticket
6. Assignee updates the resolution of the ticket as appropriate (Fixed, As Designed, etc.)
7. QA is notified of the resolution (email from Jira; state of project)
8. QA reviews the ticket resolution (when the appropriate build is available to QA)
9. QA performs one of the following actions:
 - If QA agrees with the resolution (aka, agrees that it's fixed, etc)
 - If a contractor issue (Test Track Pro' or 'TTP' in the summary): QA assign to the ticket to the reporter for contractor verification
 - Otherwise: QA closes the issue in Jira
 - QA re-opens the ticket and returns it to project management
10. QA notifies project management of the ticket status



8.7 Tracking Data Model Issues

Generic Issues identified in the database schema are to be reported in the 'Data Governance Issue Management' Jira project, using the **'Data Models' component**. Direct link is

<http://jira:8888/browse/DGIM>

Please do not report all data related issues to the above Jira project. The general rule of thumb includes:

1. This is for issues with the Data Model (table structure, view existence, etc)
2. Locate the supporting CMS spec to verify if it is tied to a data model schema release. Currently CM is grouping the specs for 3.17, etc.
 - a. If the spec is directly grouped to a schema release, then log the issue against the above Jira project
 - b. Otherwise, review the spec to determine if it is app specific, or clearly aligned to a project, or even potentially data model at a lower level (control tables?). Log issue accordingly.
 - c. If you have any questions, please discuss with QA Leadership

As always, please ensure to follow general best practices, ensuring that you include the associated specification / CMS number.

9.0 Automation

Automation is basically any script or program that makes us more efficient relative to doing the same operation manually.

9.1 Deciding what (and how) to automate

This is the million dollar question, and there isn't a single answer. In general, automate test cases that are run multiple times (leverage the investment), and the hopelessly vague "can be automated in such a way that it will take less time (or better analysis) compared to running manually".

Many of our applications have relatively simple steps, but need to be run across many different data combinations. These items are great candidates for automation.

Automating read-only applications is easier than automating add/update applications, but various SQL statements (or DataCopy) can be used to initialize data into a known state to support automation of update applications (used for MHCDP and Patient Update.)

A couple of quick introductions to Software Test Automation:

- **"An Introduction to Software Test Automation"**
http://www.qthreads.com/articles/testing/an_introduction_to_software_test_automation.html
- **Totally Data-Driven Automated Testing"** (http://sqa-test.com/w_paper1.html)

More TBD...

9.2 Different Types of Automation

9.2.1 Batch test automation

"Batch" (continuously running) testing can be used to test certain batch applications across multiple persons, and is mostly used for applications that generate 'derived' data, such as BDM and HCCMM.

The general process is to create the following:

- SQL script(s) to retrieve the derived data for x number of persons (1,000, for example.)
- A SQL script or batch file to force processing for those persons
- A way to check the status of processing
- Various directories: SQL directory, pre and post processing results directory, and documentation on testing

And the general process includes:

- Point SQL*Plus to the directory containing the appropriate SQL
- Save the data from the derived tables into a file (this is the original state.) NOTE: if time permits, it is nice to force a redetermination before moving in the new version of the application.
- Force processing

- Wait for processing to complete
- Save the data from the derived tables into a file (this is the 'after' data)
- Use a Visual Differencing tool (such as WINDIFF or WDIFF) to compare the original data with the after data. Research any differences to determine if as expected (data change) or not (potential issue with the spec or application.) Update the expected results as needed.

For an example, see K:\Qa_Testing\Apps\BDM\batch\readme.txt

This is currently used to test BDM, HCCMM, HCCMD, HCCMS, ASG_HCDP, ...

NOTE: this process is not the most effective when there are a lot of changes (many differences to analyze), but even in those situations it can be used to find crashes or "spins" (infinite loops) in the application.

9.2.2 TestPartner Test Automation

Microfocus TestPartner is a functional test automation tool that is used within DEERS QA. It has been used to automate a wide range of applications within DEERS using a data-driven / executable test matrix design.

At a high level, this type of architecture makes it easier to maintain your automation as the application changes, which often require changes to automation and the addition/changing of test cases.

The types of applications that are currently automated include:

- XML System-to-system (DCS, RBS, BBS, ...)
- DEERS TR System-to-System (Claims Coverage/CCEA, Totals Inquiry, CCD Update...)
- Web Applications (GIQD, CVS, DLPT)
- Batch CR (Continuously running) Applications (BDM, MHCDP, NtfyD, etc)
- Batch "UNIX" Automation (PFT, BEFP, CLADR Sync)
- Thick-client .Net application automation (DBIDS)
- Thick-client Java Swing application automation (RAPIDS)

Additional information on the use of automation is available from the QA SharePoint site.

9.2.3 Miscellaneous Tools / Processes

Some other tools that persons have found useful:

1. SQL to create SQL to reset the data for a family in a number of tables – an example from MHCDP to re-create enrollment data for a family (HCDP, HCDP_PLCY, HCDP_EM, TPCM_SLCT) : K:\Qa_Testing\SQLscripts_shared_readme_case_creation.docx
2. SQL to create multiple input transactions – an example from claims : K:\Qa_Testing\SQLscripts\hccmtf -- creatclaims1.SQL and creatclaimsp.SQL

9.2.4 Additional Info

http://teamsites.ds.dhra.osd.mil/teams/es/qa/QA_Automation/SitePages/Framework%20Development.aspx

9.3 "Oracles" to use in Test Automation

One of the biggest challenges in automation is defining accurate, efficient, and maintainable methods, aka "Oracles", to determine if an application behaved as expected. Per Douglas Hoffman, there are the following types of Oracles:

- “No Oracle”
- True Oracle
- Consistency
- Self Referential (SVD)
- Heuristic

Below is a Video (so CAN'T watch from government account), but a good overview of what goes into test automation, with an emphasis on strategies for result validation in test automation (which is what he refers to as an 'Oracle'.)

Using Test Oracles in Automation (NOTE: not for use at work) --

<http://video.google.com/videoplay?docid=-753663485306555503&q=Using+Test+Oracles+in+Automation>

The slides (PDF Format) : <http://www.softwarequalitymethods.com/H-Papers.html#OracleInAuto>

10.0 Monitoring Testing (AppMonitor)

QA tests functionality within 11g AppServer applications (Web/XML) that facilitate keeping a healthy environment, primarily in production, but also in other regions. This is implemented using a shared library called ‘AppMonitor’ that can check for app functionality and dependency checks against data-sources, JMS queues, directories, and work tables. Systems will use it to regularly ensure that applications are up and running in different regions.

Application testers need to incorporate AppMonitor testing into their core / build verification testing. Starting October 1, 2012, all 11g AppServer Web/XML application release need to include AppMonitor. By Feb 16th, 2013, all 11g AppServer Web/XML application need to implement AppMonitor.

AppMonitor is tested similarly to other Web testing – enter a URL (Browser, Toolkit) and check the results. The result generated by AppMonitor is an XML document. Each test and dependency that is run returns a status code: 0 for success, and 1 (or any non-zero) for failure.

To run it, add “/appMonitor.status” to your applications base URL, optionally followed by parameters. For example:

- Basic check of GIQD in Model ONE: <http://wm1.int.dmdc.osd.mil/appj/giqd/appmonitor.status/> (systems will run only the basic check in production)
- Full check of GIQD in Model ONE: <http://wm1.int.dmdc.osd.mil/appj/giqd/appmonitor.status/All> (used by various folks to research the root cause when the application is not working as expect)

A few example test cases are included in the example test case matrix (available from SharePoint.)

More information about Application Monitor can be found under:

\\Hobbes\deers\Qa_Testing\Apps\CTIS_WS\APPLICATIONS\AppMonitor and <http://teamsites.ds.dhra.osd.mil/teams/devel/SitePages/AppMonitor.aspx>

Please see the QA Assignments by Application document located on the QA SharePoint site for AppMonitor contacts in QA.

Note that given the nature of our shared environment, QA only tests ‘success’ scenarios for AppMonitor. It is up to the developer to simulate and test fail scenarios.

11.0 Security Testing

While most of us do not work with classified data, the data that we do work with is confidential.

11.1 Authentication, Authorization and Password Changes

Many applications provide multiple methods for a person to login (Authentication and Authorization), with the ability to change passwords for some of the authentication mechanisms.

Authentication can be done for both Operators and Users

Authentication schemes can use SNT, CAC, DFAS Pins, and Personal Information.

Many applications within DMDC use the DC 3 common components for authentication

Test cases that should be useable in the majority of applications for testing the various forms of authorization can be found at K:\Qa_Testing\Apps\DC3WA\testing\test_cases (Part of the DC 3 regression suite, maintained by the DC 3 primary tester.)

12.0 Testing as Part of a System (“End to End”)

Most applications at DMDC do not stand alone. Data is created, modified, and consumed by multiple applications. Most application have auditing and logging requirements. Some applications generate notifications to inform external partners of changes to the data.

It is important to understand these relationships for an application and to test as needed.

Details TBD....

12.1 Auditing & RUN / SUBM ID's

12.2 Logging

A log viewer called ‘Splunk’ is available to view the logs generated by Web Applications, and should be used at the end of testing to validate the results of the testing.

Ask your leadership on how to request access to Splunk.

User, Knowledge, & Search information: <http://docs.splunk.com/Documentation/Splunk>

12.3 Notifications

12.4 Triggers

Many tables in our databases have triggers that replicate data, validate data, set default values, force redeterminations / notifications based on a change, etc.

As QA, we need to test the triggers.

Some data on how to view the source of triggers is listed in the following section: ‘Trigger / Procedure Testing

13.0 SQL

A lot of the work that we do revolved around the data.

Some SQL statements usable across applications are documented in k:\qa_testing\SQLScripts_Shared*.doc

NOTE: SQL used within automation is kept in CVS and checked out to d:\userdata\SQL_Scripts\ -- the latest versions of SQL statements will be there.

Some oldish Oracle / Oracle SQL documentation can be found at K:\Qa_Testing\training\SQL\Oracle Reference\8.1.6_docs

Some WebSites with more information:

<http://www.orafaq.com/faqsq1.htm>

<http://www.orafaq.com/fagplus.htm>

<http://www.w3schools.com/sql/default.asp>

TBD: List common SQL statements / structure / groupings of SQL Statements.

@alt - Default settings used by many SQL scripts

@sf_xr4

@XXX_aud2 <DMDC_ID>

13.1 Some Wildcards to use with “LIKE”

SQL	Notes	Example
%	Matches any string of zero or more characters.	Find all persons with a last name containing a space: select * from pn where PN_LST_NM like '% %';
-	Matches any one character.	
[token]	Brackets enclose ranges or set, like [1-9] or [klmnopq]. 1) Range: Specify a range (e.g. [“Start” – “Stop”] 2) Set: Discrete character values in any order. [a4Bc],[abcdefg]	
[^token]	The caret “^” before a token indicates non-inclusion. Examples - [^c-g] means any character that is not a 'c', 'd', 'e', 'f' or 'g'.	

13.2 Common SQL Functions / Commands

Function	Notes	Example
COALESE		
DECODE	Oracle statement for IF-THEN-ELSE. Often used to get the appropriate PE (projected) vs TERM (actual) termination date. Or for getting discrepant vs non-discrepant data.	Find all person born on the leap day, taking discrepant data into account: SELECT DECODE(PN_VER_STAT_CD, 'D', SUSP_PN_BRTH_DT, PN_BRTH_DT) as DOB from pn Where TO_CHAR(DECODE(PN_VER_STAT_CD, 'D', SUSP_PN_BRTH_DT, PN_BRTH_DT), 'MMDD') = '0229'; More info at: http://www.techonthenet.com/oracle/functions/decode.php
NVL	Conditional processing for fields that can have	Find all PNLEC segment with a future (or null) TERM_DT.

	a null value.	select * from pnlec where NVL(PNLEC_TERM_DT, SYSDATE+100) > SYSDATE; NOTE: to be complete, should use DECODE to handle discrepant data.
UPPER	Convert a field to upper case.	Find all persons with first name 'John', regardless of case: select * from pn where UPPER(PN 1ST NM) like '%JOHN%';
SPOOL	Start saving results to a file. NOTE: cannot have a space in a directory name.	spool d:\temp\hcdp_cpf.lst select ... spool off
SPOOL OFF	Turn off spooling, save results.	
	Date related Functions	
TO_CHAR	Convert from Date to a String	
TO_DATE	Convert a String to a date.	
SYSDATE	The current date	
ADD_MONTHS	Add the specified number of months to a given date. NOTE: can give a negative number to subtract months.	Get a date three months in the future: SELECT ADD_MONTHS(SYSDATE, 3) FROM DUAL;
LAST_DAY	Retrieve that last day of the given month.	Get the last day of the current month: SELECT LAST_DAY(SYSDATE) FROM DUAL;
TRUNC	Retrieve the first day of the given month.	Get the first day of the current month: SELECT TRUNC(SYSDATE, 'MM') FROM DUAL;

13.3 SQL *Plus

SQL*Plus is the tool most often used to run interactive SQL (select, add, update.)

NOTE: when you shut-down SQL*Plus, it does an auto-commit. Ensure that you do a 'commit' or 'rollback' before shutting down SQL*Plus so that you do not accidentally delete data.

13.3.1 SQL *Plus Formatting

Function	Notes	Example
set pagesize 2000 set linesize 2000 set trimspool on set RECSEP WRAPPED set RECSEPCHAR ' '	Various command to change formatting of the results in SQL*Plus.	set pagesize 2000 set linesize 2000 set trimspool on set RECSEP WRAPPED set RECSEPCHAR ' ' set RECSEPCHAR " set linesize 100
col COLUMN_NAME format FORMAT_STRING	Change the formatting for a given field	col TABLE_NAME format A10

13.4 Trigger / Procedure Testing

A number of SQL scripts are available to view the source of triggers and procedures. These SQL statements are available from K:\Qa_Testing\SQLscripts\research\trigger

NOTE: This information is also available via JDeveloper. Info on usage at K:\Qa_Testing\training\JDeveloper

14.0 Oracle Data Dictionary

Oracle includes data dictionary views that describe the data within the database. These can be used by QA to simple checking of changes in the database. (NOTE: this is separate from any DMDC data dictionary.)

Some common views within the Oracle database (find oracle books / documentation for more information:

ALL_TABLES

ALL_VIEWS

ALL_INDEXES

ALL_TAB_COLUMNS

ALL_COL_COMMENTS

ALL_SYNONYMS

ALL_TRIGGERS

15.0 Potential Future Topics

TBD...

Appendix A: Glossary

A couple of Glossaries I have found on the web – need to pick one / incorporate info / define terms as we use them....

<http://www.aptest.com/glossary.html>

Document History

Author	Purpose	Revision	Date (ISO)
(b) (6)	First draft	0.01	2006-11-21
	Added additional information on Automation.	0.02	2006-11-27
	Added information on identifying test data.		
	Updated TOC to be clickable links		

(b) (6)	Added multiple placeholder sections (Glossary, Common Test Scenarios, "End to End" testing, SQL, ...)		
	Added in Overview section from EJ's doc; need to merge in with other overview / intro info already within the document.	0.03	2006-11-29
	Moved items to the 'Handbook' document.	0.04	2006-12-13
	Formatted Document	0.05	2006-12-22
	Minor additions on data lookup and conflict resolution	0.06	2007-07-25
	<ul style="list-style-type: none"> Updated how data tracked Updated info from WinRunner to TestPartner Updated info on issue tracking Updated common scenarios 	0.07	2011-01-24
	<ul style="list-style-type: none"> Minor enhancements to the 'Appendix B: Common Data Scenarios Needing Testing' section 	0.08	2011-04-19
	<ul style="list-style-type: none"> Incorporated comments from Mark M. Misc updates for clarification / process updates 	0.1	2011-04-26
	<ul style="list-style-type: none"> Moved 'Common Data Scenarios' section to be earlier in the document Misc review and updates 	.2	2011-08-27
	<ul style="list-style-type: none"> Incorporating review comments (Pat Rodriguez) -- Added 'Data creation tips and tricks' section; info on Photo Added 'NOTE: Max value supported: 2,147,483,647 (2³¹-1)' A couple of minor enhancements to section '3.0 Common Data Scenarios Needing Testing' 	.3	
	<ul style="list-style-type: none"> Added info in DOOR section Added several Wiki links where wiki was referenced Updated info in Logging to reflect new Splunk Logging Added several links, removed dead links Changed some K drive links to new SharePoint location. 	.4	2011-11-16
	<ul style="list-style-type: none"> Reviewed and accepted/updated Rick's changes as needed Added info on SQL script for checking DOOR Link to shared SQL for creating enrollment data Added link to DataCopy SharePoint page Misc updates as notice items while accepting revisions Fixed link on where to find template on SharePoint (for Test Plan and Test Case Matrix) Minor updates to Testing Process 	.5	2012-01-11

(b) (6)	<ul style="list-style-type: none"> Minor updates to the automation section 		
	<ul style="list-style-type: none"> Expanded the issue reporting section; based on info from the DBIDS Defect Reporting Standards 	.6	2012-09-14
	<ul style="list-style-type: none"> Added information for AppMonitor Added info on Test Case Development Minor updates to issue tracking section 	.7	2012-09-14
	<ul style="list-style-type: none"> Moved the test case development process into the Test Process' section 	.7.1	2012-09-18
	<ul style="list-style-type: none"> Section 3: 'Common Data Scenarios Needing Testing' <ul style="list-style-type: none"> Added sub-headings (easier to see what's included by looking at the TOC.) Minor updates 'Name' info Misc adding sub-headings throughout (section 5, Test Heuristics) 	.7.2	2012-11-07
	<ul style="list-style-type: none"> Added some info on the Oracle Data Dictionary Updated TOC (no revisions marks – everything shown as changed.... Added section on reporting Data model issues. 	0.7.3	2012-12-06
	<ul style="list-style-type: none"> Added a bit about application versioning 	0.7.4	2012-12-13
	<ul style="list-style-type: none"> Section 3: 'Common Data Scenarios Needing Testing' <ul style="list-style-type: none"> Added more Enrollment info 	0.8	2013-02-08